Subject: Re: Object Widgets
Posted by davidf on Sun, 07 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Bernard Puc (bpuc@va.aetc.com) writes:

> What's the current status of writing object widgets?

Alive and well. :-)

One of the difficulties with object widgets is handling the events that get generated. It is not possible, of course, to assign object methods as event handlers, so you have to have an event handler that dispatches the events to the appropriate methods.

There are several ways to do this, but one I am starting to use a lot is to use the user value of every widget that will generate an event to store the name of the method that should be called as an event handler for that widget. I write these event handler methods exactly as I do any other event handler, except that I don't have to worry about the info structure, since that is now part of the self object.

I store the self object in the user value of the top-level base (where I previously stored the info structure). This makes it possible for me to write my event handler that dispatches events to the appropriate method in exactly the same way for every object widget I write. (Note that the only case where this doesn't work is when the top-level base generates an event, I.e., a resize event. Thus, I have a special case for this.)

```
PRO MyClass__Events, event
Widget_Control, event.top, Get_UValue=selfObject
IF event.top EQ event.ID THEN thisMethod = 'RESIZE' ELSE $
  Widget_Control, event.ID, Get_UValue=thisMethod
Call_Method, thisMethod, selfObject, event
END
```

So, if I want to add a button, for example, to change the data colors, I can define the button like this:

```
colorsID = Widget_Button(colorsBase, Value='Change Colors', $
  UValue='DataColors')
```

The "DataColors" event handler method is written like this:

```
  PRO MYCLASS::DataColors, event
  thisEvent = Tag_Names(event, /Structure_Name)

  IF thisEvent EQ 'WIDGET_BUTTON' THEN BEGIN

    XColors, NColors=self.ncolors, Group_Leader=event.top, $
      NotifyID=[event.id, event.top]

  ENDIF ELSE BEGIN

      ; Must be an XCOLORS load color table event.
      ; Save color vectors and redraw on 24-bit displays.

    self.r = event.r[0:self.ncolors-1]
    self.g = event.g[0:self.ncolors-1]
    self.b = event.b[0:self.ncolors-1]
    Device, Get_Visual_Depth=thisDepth
    IF thisDepth GT 8 THEN self->Draw

 ENDELSE
 END
```

Since this event handler method is essentially identical to
the way I would write it normally, it makes it easy
to take previously written widget code and turn it into
object code.

But why bother? Well, the primary reason is that
I can make the object do things that I can't make
a widget program do. For example, with an object I
don't really *have* to have a GUI interface, unless
I want one.

I wrote a DrawColors object this weekend that acts
as a combination of my non-GUI program GetColor
and my widget program PickColor. In other words,
I create the object like this:

```
  colors = Obj_New('drawcolors')
```

If I want to load a yellow color at color table
index 200, I can do this:

```
  TVLCT, colors->GetColor("yellow"), 200
```

However, if I want the user to select a color for
color index 200 from the 16 drawing colors I have

available in a graphical user interface (or if
I want them to mix their own drawing color), I
can call the Select method, which puts up a
graphical user interface, like this:

```
thisColor = colors->Select(Cancel=cancelled)
IF NOT cancelled THEN TVLCT, thisColor, 200
```

The first case involves no GUI, the second case
does.

And if I suddenly decide that I want the sliders
in the HSV color system rather than the RBG color
system I originally designed, I can simply write
a new method for the object that changes the
labels on the sliders and the way the sliders
work. I don't have to worry about redesigning
my whole interface.

I hope this gives you some ideas. I'll probably
have the DrawColors object available on my
web page later this week after I add the program
documentation. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Object Widgets
Posted by davidf on Sun, 07 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Bernard Puc (bpuc@va.aetc.com) writes:

> What's the current status of writing object widgets?

Alive and well. :-)

One of the difficulties with object widgets is handling
the events that get generated. It is not possible, of

course, to assign object methods as event handlers, so you have to have an event handler that dispatches the events to the appropriate methods.

There are several ways to do this, but one I am starting to use a lot is to use the user value of every widget that will generate an event to store the name of the method that should be called as an event handler for that widget. I write these event handler methods exactly as I do any other event handler, except that I don't have to worry about the info structure, since that is now part of the self object.

I store the self object in the user value of the top-level base (where I previously stored the info structure). This makes it possible for me to write my event handler that dispatches events to the appropriate method in exactly the same way for every object widget I write. (Note that the only case where this doesn't work is when the top-level base generates an event, I.e., a resize event. Thus, I have a special case for this.)

```
PRO MyClass__Events, event
Widget_Control, event.top, Get_UValue=selfObject
IF event.top EQ event.ID THEN thisMethod = 'RESIZE' ELSE $
   Widget_Control, event.ID, Get_UValue=thisMethod
Call_Method, thisMethod, selfObject, event
END
```

So, if I want to add a button, for example, to change the data colors, I can define the button like this:

```
colorsID = Widget_Button(colorsBase, Value='Change Colors', $
   UValue='DataColors')
```

The "DataColors" event handler method is written like this:

```
PRO MYCLASS::DataColors, event
thisEvent = Tag_Names(event, /Structure_Name)

IF thisEvent EQ 'WIDGET_BUTTON' THEN BEGIN

   XColors, NColors=selfObject.ncolors, Group_Leader=event.top, $
      NotifyID=[event.id, event.top]

ENDIF ELSE BEGIN

      ; Must be an XCOLORS load color table event.
```

```
        ; Save color vectors and redraw on 24-bit displays.

      self.r = event.r[0:self.ncolors-1]
      self.g = event.g[0:self.ncolors-1]
      self.b = event.b[0:self.ncolors-1]
      Device, Get_Visual_Depth=thisDepth
      IF thisDepth GT 8 THEN self->Draw

  ENDELSE
  END
```

Since this event handler method is essentially identical to the way I would write it normally, it makes it easy to take previously written widget code and turn it into object code.

But why bother? Well, the primary reason is that I can make the object do things that I can't make a widget program do. For example, with an object I don't really *have* to have a GUI interface, unless I want one.

I wrote a DrawColors object this weekend that acts as a combination of my non-GUI program GetColor and my widget program PickColor. In other words, I create the object like this:

```
   colors = Obj_New('drawcolors')
```

If I want to load a yellow color at color table index 200, I can do this:

```
   TVLCT, colors->GetColor("yellow"), 200
```

However, if I want the user to select a color for color index 200 from the 16 drawing colors I have available in a graphical user interface (or if I want them to mix their own drawing color), I can call the Select method, which puts up a graphical user interface, like this:

```
   thisColor = colors->Select(Cancel=cancelled)
   IF NOT cancelled THEN TVLCT, thisColor, 200
```

The first case involves no GUI, the second case does.

And if I suddenly decide that I want the sliders

in the HSV color system rather than the RBG color
system I originally designed, I can simply write
a new method for the object that changes the
labels on the sliders and the way the sliders
work. I don't have to worry about redesigning
my whole interface.

I hope this gives you some ideas. I'll probably
have the DrawColors object available on my
web page later this week after I add the program
documentation. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Object Widgets
Posted by davidf on Mon, 08 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Pavel Romashkin (promashkin@cmdl.noaa.gov) writes:

> Thank you David. This was great. Did you include this on your web site?
> Please do.

I'm working on it this morning. Documentation, of course,
takes about 10 times as long as writing the program. :-(
It should be done later today or tomorrow.

> I wonder if RSI is doing anything of the sort.

Oh, I'm sure they are. In fact, I can't wait to see it. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting

Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Object Widgets
Posted by Pavel Romashkin on Mon, 08 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Thank you David. This was great. Did you include this on your web site?
Please do. I wonder if RSI is doing anything of the sort.

Cheers,
Pavel

Subject: Re: Object Widgets
Posted by Struan Gray on Mon, 08 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

David Fanning, davidf@dfanning.com writes

> One of the difficulties with object widgets is handling
> the events that get generated. It is not possible, of
> course, to assign object methods as event handlers,
> so you have to have an event handler that dispatches
> the events to the appropriate methods.

I have found that I can have just one such event handler routine in an ur-widget, which automatically calls the correct event handler of subclassed widgets. All my 'real' widgets need is a structure definition which inherits the ur-widget, an INIT method and an event handling method.

I am actually in the process of putting together web pages to describe the widget and how to effectively subclass other widgets from it, but as always other things keep getting in the way. If people want to look at the code, the following pages has links to the ur-widget, a minimal widget which actually does something, and shows how to inherit from and interact with the ur-widget, and a more complex colour-picker widget.

http://www.sljus.lu.se/stm/IDL/Obj_Widgets/

I promise, hand on heart, to try and get some explanatory text written up real soon now. I'm now using this ur-widget for all my standalone widgets (I've not started trying to make any compound

widgets with it, though I don't see why it wouldn't work), and love
the way I can add functionality to the ur-widget (see the SHOW method)
which is then immediately available to all my other widgets, including
those up and running on the screen.

Struan

Subject: Re: Object Widgets
Posted by Struan Gray on Wed, 10 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Mark Hadfield, m.hadfield@niwa.cri.nz writes:

>> http://www.sljus.lu.se/stm/IDL/Obj_Widgets/
>
> Interesting stuff, Struan! I note that your widget
> template object, SLFoWid, does not create a top-level base.
> This seems to me to be an obvious thing to do, since all
> widget hierarchies require a TLB. What were the reasons for
> your choice? Do you expect SLFoWid to be used for widgets
> that aren't at the top level?

That's the main idea. I want SLFoWid to define behaviour, not
layout, and I would like to use it for compound widgets. This is why
I have a seperate SLFOWID::XMANAGE method. I also found that putting
all the widget creation statements into one INIT method makes it
easier to maintain the code, and avoids having to pass around all the
possible keywords to WIDGET_BASE. The web pages will (eventually)
contain some discussion of my design decisions.

I think my news server is behind the times a bit, as I got an
email from you containing the following good points (I've changed
their order). If you didn't post them as well, sorry for the breach
of netiquette.

> 2. Similarly it is a good idea in the object cleanup
> routine to check if the widget hierarchy is still valid and,
> if it is, destroy it, i.e.
>
> pro SLFoWid::Cleanup
>
> print, 'SLFoWid::Cleanup'
> print, ' widget ID: ', self.myWidID
> print, ' object ID: ', self
>

> if widget_info(self.myWidID, /VALID_ID) then $
> widget_control, self.myWidID, /DESTROY
>
> end ; pro SLFoWid::Cleanup

Strangely enough, despite promoting widgets-as-objects I had assumed that the widget would always be killed with WIDGET_CONTROL - which is daft I admit. I wanted to avoid getting stuck in an infinite loop, with the widget and object cleanup routines calling each other over and over. One way to avoid that is for the object cleanup to use WIDGET_INFO to see if self.myWidID is being managed, but that assumes the Xmanager is being used. Another is for the object and widget cleanup routines to use keywords when they call each other, but that seems inelegant.

After a little digging it seems that both WIDGET_CONTROL, /DESTROY and OBJ_DESTROY take care to avoid recursion - ie, they seem to know that things are in the process of being destroyed, despite the fact that both the widget and the object IDs are still valid in their own cleanup routines. (note that your 'if widget_info...' is redundent, since self.myWidID is valid until all the cleanup is done). Since WIDGET_CONTROL and OBJ_DESTROY seem to ignore demands to kill widgets/objects that they're already in the process of killing, I've just added a WIDGET_CONTROL, /DESTROY line to the object cleanup method.

If RSI allows us to specify an object method as the cleanup routine for a widget this problem will disappear.

> 1. In your widget cleanup routine it is a good idea to
> check that the object is valid before trying to destroy it,
> i.e.
>
> pro SLFoWid_Cleanup, myID
>
> widget_control, myID, get_uvalue=myObjRef
> if obj_valid(myObjRef) then begin
> myObjRef -> GetProperty, no_block=no_block
> if no_block eq 1 then obj_destroy, myObjRef $
> else myObjRef -> cleanup
> endif
>
> end ; pro SLFoWid_Cleanup
>
> This is advisable because it is possible for the object
> to have been destroyed (by OBJ_DESTROY, or HEAP_GC) behind

> XMANAGER's back.

As I said above, I was assuming that the widget would always be destroyed as a widget, not as an object. That said, even after the change to the object cleanup routine above, there is no need to check the object reference for validity since the only way you can end up in an object or widget cleanup routine is when it dies, and at that point the widget ID and object ID must still be valid. I've left your check in all the same: an inattentive programmer may have stored something else in the uvalue and lost the object reference. The check would still leave a dangling object reference, but at least the program won't crash.

> 3. There is a problem in heap cleanup for blocking widgets.
> If I create anew widget with
>
> o = obj_new('SLFow_minimal')
>
> then hit the quit button, the heap is cleaned up. But if
> I do the same with
>
> o = obj_new('SLFow_minimal', /BLOCK)
>
> then the 'SLFow_minimal' object is left on the heap

This one I did catch myself (but hadn't updated the web version). I saw your later post, which uses the same solution I have, though I use this line:

return, self.no_block

at the end of all the INIT methods of subclassed widgets. In general, I wanted to minimise the amount of stuff a subclass writer would have to remember to include in their methods, but this seems essential (as is getting the _ref_extra and _extra keywords right).

Note that obj_destroy called on a blocking widget (from another widget, or from a 'quit' button) will always produce an error, so it is safest to use widget_control, /destroy if you don't know beforehand whether a widget will be used in blocking or non-blocking mode. Actually, I can see few reasons to use blocking widgets, but since I thought I could handle them it seemed churlish to force widgets to be non-blocking.

Thanks for the tips and comments.

Struan

Subject: Re: Object Widgets
Posted by Mark Hadfield on Wed, 10 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

More on Struan's widgets. Sorry for thinking aloud on the group but I find it all quite interesting.

I have thought of a simple (and in hindsight obvious) solution for the heap cleanup problem for blocking widgets. Just have the Init method for blocking widgets always return 0. That way the object exists (in some sense) while the widget application is running but when the widget application exits it returns to the Init method, which tells IDL that initialisation was not successful. Sounds dodgy, but seems to work! So the Init method in Struan's code becomes (additions in capitals):

```
function SLFow_minimal::init, BLOCK=BLOCK,  _ref_extra=extra

  if not (self->SLFoWid::init(BLOCK=BLOCK, _extra=extra)) then begin
     print, 'SLFow_minimal: failed to initialise SLFoWid'
     return, 0
  endif

  self.myWidID = widget_base(uvalue=self, title=self.title, /column,
xsize=200)
  temp = widget_button(self.myWidID, value='Undefined')
  self.quitbut = widget_button(self.myWidID, value='Quit')

  widget_control, self.myWidID, /realize

  self -> xmanage

  IF KEYWORD_SET(BLOCK) THEN RETURN, 0 ELSE return, 1

end   ; function SLFow_minimal::init
```

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: Object Widgets
Posted by Mark Hadfield on Wed, 10 Nov 1999 08:00:00 GMT

A few more items of feedback on your widget classes, Struan:

1. In your widget cleanup routine it is a good idea to check that the object is valid before trying to destroy it, i.e.

```
pro SLFoWid_Cleanup, myID

  widget_control, myID, get_uvalue=myObjRef
  if obj_valid(myObjRef) then begin
    myObjRef -> GetProperty, no_block=no_block
    if no_block eq 1 then obj_destroy, myObjRef $
      else myObjRef -> cleanup
  endif

end   ; pro SLFoWid_Cleanup
```

This is advisable because it is possible for the object to have been destroyed (by OBJ_DESTROY, or HEAP_GC) behind XMANAGER's back.

2. Similarly it is a good idea in the object cleanup routine to check if the widget hierarchy is still valid and, if it is, destroy it, i.e.

```
pro SLFoWid::Cleanup

  print, 'SLFoWid::Cleanup'
  print, '  widget ID: ', self.myWidID
  print, '  object ID: ', self

  if widget_info(self.myWidID, /VALID_ID) then widget_control, self.myWidID,
/DESTROY

end   ; pro SLFoWid::Cleanup
```

3. There is a problem in heap cleanup for blocking widgets. If I create a new widget with

```
o = obj_new('SLFow_minimal')
```

then hit the quit button, the heap is cleaned up. But if I do the same with

```
o = obj_new('SLFow_minimal', /BLOCK)
```

then the 'SLFow_minimal' object is left on the heap. The problem, as you have noted in comments in your code, is that for a blocking widget the procedure that handles the 'quit' event is called from the XMANAGER event loop, which is called from inside the Init method, and you can't destroy an object from inside it's own Init method. One solution is for

SLFow_minimal::Init to check whether it has been called with the BLOCK keyword set. If it has, then it needs to leave out the call to Xmanage, and leave this up to the user. This is an unfortunate complication. I guess the other solution is for 'SLFow_minimal' to prevent the BLOCK keyword from being passed to 'SLFoWid' so that the application always runs non-blocking. (Perhaps you'd already thought of all this.)

Anyway, thanks very much for publishing this stuff. It provides a framework for writing much cleaner widget applications.

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: Object Widgets
Posted by Mark Hadfield on Wed, 10 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Struan Gray <struan.gray@sljus.lu.se> wrote in message
news:8064k5kph1@news.lth.se...
> ...
>   ...the following pages has links to the
> ur-widget, a minimal widget which actually does something, and shows
> how to inherit from and interact with the ur-widget, and a more
> complex colour-picker widget.
>
>   http://www.sljus.lu.se/stm/IDL/Obj_Widgets/

Interesting stuff, Struan! I note that your widget template object, SLFoWid, does not create a top-level base. This seems to me to be an obvious thing to do, since all widget hierarchies require a TLB. What were the reasons for your choice? Do you expect SLFoWid to be used for widgets that aren't at the top level?

Mark Hadfield
m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

Subject: Re: Object Widgets
Posted by J.D. Smith on Wed, 10 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Struan Gray wrote:
>
> Mark Hadfield, m.hadfield@niwa.cri.nz writes:
>
>>> http://www.sljus.lu.se/stm/IDL/Obj_Widgets/
>>
>> Interesting stuff, Struan! I note that your widget
>> template object, SLFoWid, does not create a top-level base.
>> This seems to me to be an obvious thing to do, since all
>> widget hierarchies require a TLB. What were the reasons for
>> your choice? Do you expect SLFoWid to be used for widgets
>> that aren't at the top level?
>
> That's the main idea. I want SLFoWid to define behaviour, not
> layout, and I would like to use it for compound widgets. This is why
> I have a seperate SLFOWID::XMANAGE method. I also found that putting
> all the widget creation statements into one INIT method makes it
> easier to maintain the code, and avoids having to pass around all the
> possible keywords to WIDGET_BASE. The web pages will (eventually)
> contain some discussion of my design decisions.
>
> I think my news server is behind the times a bit, as I got an
> email from you containing the following good points (I've changed
> their order). If you didn't post them as well, sorry for the breach
> of netiquette.
>
>> 2. Similarly it is a good idea in the object cleanup
>> routine to check if the widget hierarchy is still valid and,
>> if it is, destroy it, i.e.
>>
>> pro SLFoWid::Cleanup
>>
>> print, 'SLFoWid::Cleanup'
>> print, ' widget ID: ', self.myWidID
>> print, ' object ID: ', self
>>
>> if widget_info(self.myWidID, /VALID_ID) then $
>> widget_control, self.myWidID, /DESTROY
>>
>> end ; pro SLFoWid::Cleanup
>
> Strangely enough, despite promoting widgets-as-objects I had
> assumed that the widget would always be killed with WIDGET_CONTROL -
> which is daft I admit. I wanted to avoid getting stuck in an infinite
> loop, with the widget and object cleanup routines calling each other
> over and over. One way to avoid that is for the object cleanup to use
> WIDGET_INFO to see if self.myWidID is being managed, but that assumes
> the Xmanager is being used. Another is for the object and widget

> cleanup routines to use keywords when they call each other, but that
> seems inelegant.
>
> After a little digging it seems that both WIDGET_CONTROL, /DESTROY
> and OBJ_DESTROY take care to avoid recursion - ie, they seem to know
> that things are in the process of being destroyed, despite the fact
> that both the widget and the object IDs are still valid in their own
> cleanup routines. (note that your 'if widget_info...' is redundent,
> since self.myWidID is valid until all the cleanup is done). Since
> WIDGET_CONTROL and OBJ_DESTROY seem to ignore demands to kill
> widgets/objects that they're already in the process of killing, I've
> just added a WIDGET_CONTROL, /DESTROY line to the object cleanup
> method.
>
> If RSI allows us to specify an object method as the cleanup
> routine for a widget this problem will disappear.
>
>> 1. In your widget cleanup routine it is a good idea to
>> check that the object is valid before trying to destroy it,
>> i.e.
>>
>> pro SLFoWid_Cleanup, myID
>>
>> widget_control, myID, get_uvalue=myObjRef
>> if obj_valid(myObjRef) then begin
>> myObjRef -> GetProperty, no_block=no_block
>> if no_block eq 1 then obj_destroy, myObjRef $
>> else myObjRef -> cleanup
>> endif
>>
>> end ; pro SLFoWid_Cleanup
>>
>> This is advisable because it is possible for the object
>> to have been destroyed (by OBJ_DESTROY, or HEAP_GC) behind
>> XMANAGER's back.
>
> As I said above, I was assuming that the widget would always be
> destroyed as a widget, not as an object. That said, even after the
> change to the object cleanup routine above, there is no need to check
> the object reference for validity since the only way you can end up in
> an object or widget cleanup routine is when it dies, and at that point
> the widget ID and object ID must still be valid. I've left your check
> in all the same: an inattentive programmer may have stored something
> else in the uvalue and lost the object reference. The check would
> still leave a dangling object reference, but at least the program
> won't crash.
>
>> 3. There is a problem in heap cleanup for blocking widgets.

>> If I create anew widget with
>>
>>   o = obj_new('SLFow_minimal')
>>
>> then hit the quit button, the heap is cleaned up. But if
>> I do the same with
>>
>>   o = obj_new('SLFow_minimal', /BLOCK)
>>
>> then the 'SLFow_minimal' object is left on the heap
>
>   This one I did catch myself (but hadn't updated the web version).
> I saw your later post, which uses the same solution I have, though I
> use this line:
>
>   return, self.no_block
>
>   at the end of all the INIT methods of subclassed widgets. In
> general, I wanted to minimise the amount of stuff a subclass writer
> would have to remember to include in their methods, but this seems
> essential (as is getting the _ref_extra and _extra keywords right).
>
>   Note that obj_destroy called on a blocking widget (from another
> widget, or from a 'quit' button) will always produce an error, so it
> is safest to use widget_control, /destroy if you don't know beforehand
> whether a widget will be used in blocking or non-blocking mode.
> Actually, I can see few reasons to use blocking widgets, but since I
> thought I could handle them it seemed churlish to force widgets to be
> non-blocking.

I have a similar though less tightly coupled object widget superclass. One suggestion I had would be to register the widget under the name:

obj_class(self)+self.title

or some such, in order to allow different instances of the same class to run at once. Another suggestion involves blocking vs. non-blocking widgets. I find it convenient to make a separate "Start" method in addition to Init. Init is used to set up data, etc., and Start is used to actually run XManager, etc. Though this forces the user to remember on more thing, it's actually useful in other contexts as well... especially if several widgets programs which will be interacting are being created. It has a similar flavor to XManager's JUST_REGISTER keyword.

JD

--
J.D. Smith |*| WORK: (607) 255-5842

Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|

Subject: Re: Object Widgets
Posted by Struan Gray on Thu, 11 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Struan Gray, struan.gray@sljus.lu.se writes:

Whoops. Bad formatting in my last post.

This:

> objwidref = obj_new('object_widget') objwidref -> Start

Should read:

> objwidref = obj_new('object_widget')
> objwidref -> Start

Struan

Subject: Re: Object Widgets
Posted by Struan Gray on Thu, 11 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

J.D. Smith, jdsmith@astro.cornell.edu writes:

> I have a similar though less tightly coupled object
> widget superclass. One suggestion I had would be to
> register the widget under the name:
>
> obj_class(self)+self.title
>
> or some such, in order to allow different instances
> of the same class to run at once.

The xmanger allows duplicate names, so multiple instances of a given sub-widget can exist in my version (eg, different widgets can simultaneously have their own colour pickers on the screen). Registering with the title as well would allow more sophisticated use of the XREGISTERED function, but that returns so little information that I found it only useful for singleton widgets (for setting global

preferences etc).

I'm sure J.D. spotted this, but it's worth pointing out to the novice objecters that using the obj_class(self) in the call to xmanager ensures that the subclass name is used, even if the call is made in the super-class init method.

Incidentally, for a long time I thought the Xmanager actually needed to know the actual name of the creation routine for the widget so that it could call it at some point. It seems as if it will actually accept any old name, although default names for the cleanup and event handler are of course derived from that.

> Another suggestion involves blocking vs. non-blocking
> widgets. I find it convenient to make a separate "Start"
> method in addition to Init. Init is used to set up data,
> etc., and Start is used to actually run XManager, etc.

In a sense, that's what my XMANAGE method is. Other than starting the Xmanager, is there anything else you do for all your widgets that would be worth putting into a superclass defintion? Or is it just that it is useful later on if all users know that object widgets have to he invoked with:

objwidref = obj_new('object_widget') objwidref -> Start

instead of just the first line. I can see how this solves the blocking widget/xmanager dilemma.

Thanks for the input. I begin to see why David loves having his code rubbished in public :-)

Struan
